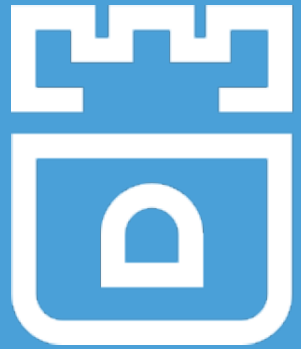

Rook Deep Dive

Travis Nielsen
Rook Senior Maintainer
Red Hat

https://rook.io/
https://github.com/rook/rook

What is Rook?

- Open Source Storage Control Plane for Kubernetes
  - Extends Kubernetes with CRDs and operators for each storage provider
- CNCF Incubation project
- Automates deployment, bootstrapping, configuration, provisioning, scaling, upgrading, migration, disaster recovery, monitoring, and resource management

Rook Operators

- Implements the **Operator Pattern** for storage solutions
- User defines *desired state* for the storage cluster
- The Operator runs reconciliation loops
  - Observe - Watch for changes in state
  - Analyze - Determine differences to apply
  - Act - Apply changes to the cluster

Custom Resource Definitions (CRDs)

- Arbitrary types that extend the Kubernetes API
  - look just like any other built-in object (e.g. Pod)
  - Enabled native `kubectl` experience
- A means for user to describe their desired state

Rook Framework for Storage Solutions

- Rook is more than just a collection of Operators and CRDs
- **Framework** for storage providers to integrate their solutions into cloud-native environments
  - Storage resource normalization
  - Operator patterns/plumbing
  - Common policies, specs, logic
  - Testing effort
- Ceph, CockroachDB, Minio, NFS, Cassandra, Nexenta

Storage Providers

- Data Platforms
  - Ceph, Minio, NFS, Nexenta
- Databases
  - CockroachDB, Cassandra
- “Adding a new Storage Provider to Rook”
  - 4:30, Room 611: Jared Watts, Upbound

Ceph Deep Dive

General Orchestration Approach

- Operator runs ceph commands to initialize and bootstrap cluster (cephx auth, crush map, etc.)
- Operator creates `Deployments` to manage the lifecycle of each Ceph daemon
  - Init containers will generate the Ceph config
  - Ceph daemons will run directly in the main container
- Health of cluster and components is monitored over time and corrective actions taken

CephCluster

- Ceph CRDs are v1
- Host path
  - Required for persistence of the data
- Dashboard
  - Web UI to view and manage the Ceph cluster
- Network

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  name: my-cluster
spec:
  dataDirHostPath: /var/lib/rook
  cephVersion:
    Image: ceph/ceph:v13.2.2-20181023
  dashboard:
    enabled: true
  network:
    hostNetwork: true
  ...
```

CephCluster: Version

- Independent from the Rook version
- Controls the data plane
- Updated independently from the Rook version
- Support lifetime depends on the Ceph project
- https://hub.docker.com/r/ceph/ceph/tags/

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  name: my-cluster
spec:
  ...
  cephVersion:
    Image: ceph/ceph:v13.2.2-20181023
```

Ceph Version Differences

- Luminous (v12)
  - Readonly dashboard running on **http** port 80
- Mimic (v13)
  - Configurable dashboard running on **https** port 8443
  - Generate a self-signed cert and admin password for the dashboard
- Nautilus (master): `allowedUnsupported: true`
  - Secure dashboard similar to Mimic
  - Enables the orchestrator mgr modules

Automated Stateful Upgrades

- Mostly Automated in v0.9
- Operator controls and manages software upgrade flow
- Upgrade is simply applying/reconciling desired state
- Leverages built-in functionality of K8s resources like `Deployments` to update components in a rolling fashion
- Separation of Rook and Ceph versioning to isolate impact

Rook Upgrades

- Updates the Rook operator and related orchestration components
- Does **not** update the data path
- Set the new version in the operator deployment

```
kubectl -n rook-ceph-system set image \
  deploy/rook-ceph-operator rook-ceph-operator=rook/ceph:v0.9.0
```

Ceph Upgrades

- Updates the data path
  - Luminous, Mimic, or Nautilus
- Set the new version in the CephCluster CRD
- Special upgrade and migration steps between major versions of Ceph (Mimic -> Nautilus) will be implemented as necessary

DEMO: Upgrades

- Configure Ceph Luminous
- Update Ceph from Luminous to Mimic

CephCluster: Mon Settings

- Mons must maintain quorum
  - Paxos: majority is necessary
- Recommended settings
  - 3 mons: Tolerant of single node failure
  - Place on unique nodes

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  name: my-cluster
spec:
  ...
  mon:
    count: 3
    multiPerNode: false
```

Orchestration of Monitors

- `Deployment` object wraps each mon pod for reliable lifecycle management
- `Service` object is created per mon to establish a consistent IP address - important for quorum and mon map

```
apiVersion:
  ceph.rook.io/v1beta1
kind: Cluster
metadata:
  name: my-cluster
spec:
  mon:
    count: 3
    multiPerNode: false
```

Monitors: Maintaining quorum

- Mon quorum is critical to cluster health
- Operator regularly checks on mon quorum
- If a mon falls out of quorum for too long, the operator takes action to replace the failed mon
  - A new mon is started (new `Deployment` and `Service` IP)
  - Wait for new mon to join quorum
  - Delete the failed mon `Deployment` and `Service`
  - Remove the failed mon from the mon map

Orchestration of OSDs

- Operator starts OSDs according to config from Cluster CRD
- Operator schedules a `Job` on each node to initialize/provision its OSDs
- One `Deployment` is created for each OSD
  - OSDs run independently
- Horizontal scaling: Operator automatically adds OSDs to new nodes and devices

CephCluster: OSD Settings

- Automatic selection mode
  - Rook will discover available devices on all nodes
  - Configure OSDs on all devices that are not in use

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  name: auto-cluster
spec:
  ...
  storage:
    useAllNodes: true
    useAllDevices: true
```

CephCluster: OSD Settings

- Specific selection mode
  - Rook will only use the nodes specified
  - OSDs will only be configured on the devices specified, if available

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  name: specific-cluster
spec:
  ...
  storage:
    useAllNodes: false
    useAllDevices: false
    nodes:
    - name: node1
      devices:
      - name: sda
      - name: sdb
```

CephCluster: OSD Settings

- Filtering mode
  - Rook will discover available devices on all nodes
  - Configure OSDs on all devices that match the filter

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  name: filter-cluster
spec:
  ...
  storage:
    useAllNodes: true
    useAllDevices: false
    deviceFilter: ^sd.
```

CephCluster: OSD Settings

- Node Labels
- Rook will only place OSDs on nodes that match the node filter
- role=storage-node

```
storage:
  useAllNodes: true
  useAllDevices: true
placement:
  osd:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: role
            operator: In
            values:
            - storage-node
```

CephCluster: OSD Settings

- Performance Optimization
  - MetadataDevice (SSD/NMMe):
    - Bluestore: WAL and DB
    - Filestore: Journal
  - Data stored on other devices (HDDs)

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  name: perf-cluster
spec:
  ...
  storage:
    useAllNodes: true
    useAllDevices: false
    deviceFilter: ^sd.
    config:
      metadataDevice: nvme01
```

CephCluster: OSD Settings

- High Performance
  - If NVMe devices are available for data, create multiple OSDs per device
  - Compute requirements are more than storage overhead

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  Name: high-perf-cluster
spec:
  ...
  storage:
    useAllNodes: true
    useAllDevices: false
    deviceFilter: ^nvme.
    config:
      osdsPerDevice: 5
```

Orchestration of RBD Mirroring

- Creates a `Deployment` for each RBD Mirroring worker
- Configure which pools and block images are to be mirrored with the Rook toolbox

```
apiVersion:
  ceph.rook.io/v1
kind: CephCluster
metadata:
  Name: high-perf-cluster
spec:
  ...
  rbdMirroring:
    workers: 3
```

Orchestration of RGW

- Creates an object gateway according to settings in the `ObjectStore` CRD
- Required Ceph pools are created
  - 5 metadata pools
  - 1 data pool (can be erasure coded)
- RGW pods are started via `Deployment` for HA/reliability
- `Service` created for client access and load balancing

```
apiVersion:
ceph.rook.io/v1beta1
kind: ObjectStore
metadata:
  name: my-store
spec:
  metadataPool:
    replicated:
      size: 3
  dataPool:
    erasureCoded:
      dataChunks: 2
      codingChunks: 2
  gateway:
      port: 80
      instances: 1
```

Orchestration of CephFS

- Creates a shared file system according to settings in the `Filesystem` CRD
- Required Ceph pools are created
  - 1 metadata pool
  - 1 data pool (can be erasure coded)
- MDS pods are started via `Deployment` for HA/reliability
  - Standby MDS pods for quick failover

```
apiVersion:
ceph.rook.io/v1beta1
kind: Filesystem
metadata:
  name: my-filesystem
spec:
  metadataPool:
    replicated:
      size: 3
  dataPools:
    - replicated:
        size: 3
  metadataServer:
    activeCount: 1
    activeStandby: true
```

Rook Agent

- Dynamically attaches/mounts Ceph storage for pod consumption
- Runs as `DaemonSet` on all schedulable nodes in cluster
- Block: `rbd map`
- File: `mount -t ceph`
- Fencing and locking for ReadWriteOnce
- Detach and reattach if pod scheduled onto another node
- Currently a Kubernetes FlexVolume
- Will be replaced by CSI driver in the near future

How to get involved?

- Rook Booth
- Contribute to Rook
  - https://github.com/rook/rook
  - https://rook.io/
- Slack - https://rook-io.slack.com/
  - #conferences now for Kubecon China
- Twitter - @rook_io
- Forums - https://groups.google.com/forum/#!forum/rook-dev
- Community Meetings

Questions?

https://github.com/rook/rook

https://rook.io/

Thank you!

https://github.com/rook/rook

https://rook.io/